Cambridge University Engineering Department
Engineering Tripos Part IIA
PROJECTS: Interim and Final Report Coversheet

IIA Projects

TO BE COMPLETED BY THE STUDENT(S)

Project:	SF4 – Data Logger	
Title of report:	Group Report	
Name(s): (capitals)	crsID(s):	College(s):
WILL HEWES	Wh365	Pembroke
CHENG FANG	Cf573	Robinson

Declaration for: Interim Report 1

We confirm that, except where indicated, the work contained in this report is our own original work.

Instructions to markers of Part IIA project reports:

Grading scheme

Grade	A*	A	B	C	D	E
Standard	Excellent	Very Good	Good	Acceptable	Minimum acceptable for Honours	Below Honours

Grade the reports by ticking the appropriate guideline assessment box below, and provide feedback against as many of the criteria as are applicable (or add your own). Feedback is particularly important for work graded C-E. Students should be aware that different projects and reports will require different characteristics.

Penalties for lateness: Interim Reports: 3 marks per weekday; Final Reports: 0 marks awarded – late reports not accepted.

Guideline assessment (tick one box)

A*/A	**A/B**	**B/C**	**C/D**	**D/E**

Marker:		Date:	

Delete (1) or (2) as appropriate (for marking in hard copy – different arrangements apply for feedback on Moodle):

(1) Feedback from the marker is provided on the report itself.

(2) Feedback from the marker is provided on second page of cover sheet.

	Typical Criteria	Feedback comments
Project Skills, Initiative, Originality	Appreciation of problem, and development of ideas	
	Competence in planning and record-keeping	
	Practical skill, theoretical work, programming	
	Evidence of originality, innovation, wider reading (with full referencing), or additional research	
	Initiative, and level of supervision required	
Report	Overall planning and layout, within set page limit	
	Clarity of introductory overview and conclusions	
	Logical account of work, clarity in discussion of main issues	
	Technical understanding, competence and accuracy	
	Quality of language, readability, full referencing of papers and other sources	
	Clarity of figures, graphs and tables, with captions and full referencing in text	

ENGINEERING TRIPOS PART IIA

SF4 – DataLogger

First Interim Report

Cheng Fang – cf573
Robinson College

Will Hewes – wh365
Pembroke College

Contents

1 Introduction

This project aims to develop a microcontroller-based automatic plant watering system. The system will autonomously monitor soil moisture levels, plot the data over time, and provide options to water the plant at scheduled intervals or in response to threshold moisture levels. This allows for effective monitoring and care with minimal user intervention.

In addition to moisture sensing, the system could be expanded to incorporate light and temperature sensors, enabling further environmental data to be plotted and analysed. These additional features enhance the system's utility for controlling the conditions affecting plant health.

Cheng will be taking the lead on the circuit design and firmware, and Will will be taking the lead on the software and hardware.

2 Parts List

Table 1 in the Appendix displays the components ordered to supplement the project so far. Their usages are discussed further in sections 3 and 4.

The implementation of the water valve in this project is based on an open-source design available online [6], which utilizes a servo-powered pinch valve mechanism.

3 Analogue Circuit Design

Figure 1 in the Appendix shows the analogue circuit design for the automatic watering system.

The 5V wire acts as a supply rail, connecting to each of the components and routing the ground through where necessary, while the I/O pins map to their corresponding components. The 5V supply voltage lies within the operating range of all connected sensors, enabling direct connection to the Arduino without the need for additional filters or amplifiers, as the sensor outputs fall within the Arduino's ADC input range and provide sufficient signal strength and stability [1]. Capacitors are added in parallel with each of the sensors to reduce sensitivity to noise, allowing the data to be tracked more accurately and eliminating some of the interference.

In designing the analogue circuit, we referred to the official documentation [1, 2, 4] of each sensor and actuator, as well as guidelines provided on the Arduino platform. For example, the TMP36 temperature sensor datasheet recommends placing a 0.1 µF ceramic bypass capacitor close to the supply pin to suppress high-frequency interference and prevent DC output shifts due to radio frequency noise [2]. Similarly, a 100 µF capacitor is added across the power supply to the servo motor, as recommended in Arduino documentation, to smooth out voltage fluctuations caused by sudden changes in current draw during operation [1].

Once the sensors are received and tested, additional analogue circuitry may be introduced depending on their real-world performance. For instance, if the soil moisture sensor exhibits limited precision or slow response time, measures such as adding a low-pass filter to stabilise fluctuating readings, using an operational amplifier to improve signal resolution, or incorporating a comparator circuit to implement threshold-based triggering could be considered [4, 5]. These enhancements would help ensure more reliable and responsive behaviour, particularly in applications requiring real-time monitoring or control.

4 Block Diagram

Figure 2 shown in the Appendix displays the general mode of operation for the system. The sensor modules send real-time information to the Arduino, which processes and transmits the data to the PC via USB serial communication.

This data is then stored by the PC and displayed graphically with Python, with an interactive GUI through which the theshold values, watering timings, or manual watering options can be controlled by user input.

This is then fed back to the Arduino, which will either do nothing if water levels are above the threshold, or dispense a controlled quantity of water by activating the servo if moisture levels have fallen below the threshold (or manual watering is toggled).

The Arduino will also monitor temperature levels and can be expanded to incorporate light sensors as well, allowing for increased datalogging capabilities.

5 Firmware Design

The firmware was developed in Arduino C/C++ based on the current circuit design. It is responsible for real-time sampling of sensor data, including the TMP36 temperature sensor and the capacitive soil moisture sensor. To improve robustness and reduce the effect of noise, the firmware performs averaging across multiple samples (10 samples by default) for each sensor before converting the raw ADC values into meaningful physical quantities—Celsius for temperature and a raw moisture scale from the soil sensor.

After processing, the firmware sends the results to the computer via serial communication every second in a simple, human-readable format (e.g., `T=24.5,M=312`). In addition, the firmware listens for control commands from the computer to drive the servo motor, which actuates the watering mechanism. The servo angle can be controlled by commands such as `SERVO=90`.

The firmware is initially based on and modified from example codes provided by the official documentation for each sensor and actuator:

- TMP36 temperature sensor code was adapted from Analog Devices' official documentation [2].
- Soil moisture sensor code is adapted from DFRobot's official Arduino example [4].
- Servo motor control follows the standard Arduino Servo library example [1].

All logic and structure have been customized to meet the needs of this system while maintaining clarity and modularity for future expansion, such as adding digital filters or handling more sensors.

6 Software Design

So far, the software design is in its early stages, with a mock-up demonstration shown in the Appendix. The final software implementation will feature Arduino communication to the PC via the serial, plotting capabilities and a GUI, and an Arduino controlled watering mechanism.

The GUI will be implemented through the Tkinter module in Python, storing data in a .csv file

7 Conclusion

The project will work towards building an automated watering system, capable of logging data with regard to soil moisture levels and temperature. The key components have been ordered, and a preliminary block diagram and circuit design have been constructed - though they are subject to change as we test the component properties.

A mock-up software design has been created which allows us to simulate an Arduino serial communication and display the current values. This will be further progressed by implementing Arduino communication once the sensors are implemented, and by enabling plotting, data storage capabilities, and an interactive GUI.

A Parts List

Order Code	Description of Component	Qty	Unit Price (£)	Total Price (£)
2946124	Capacitive Soil Moisture Sensor Module	1	4.69	4.69
SC21096	Mini servo	1	2.94	2.94
4030054	Temperature sensor	1	1.38	1.38

Table 1: Component Order Summary

B Analogue Circuit Design

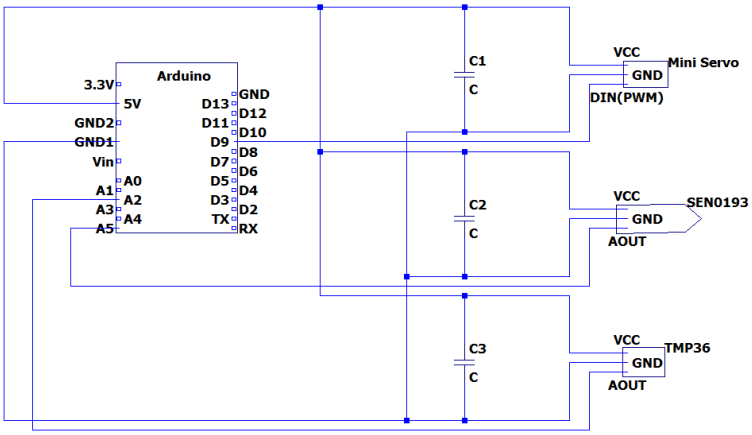

Figure 1: Analogue Circuit Diagram for the automatic watering system

C Block Diagram

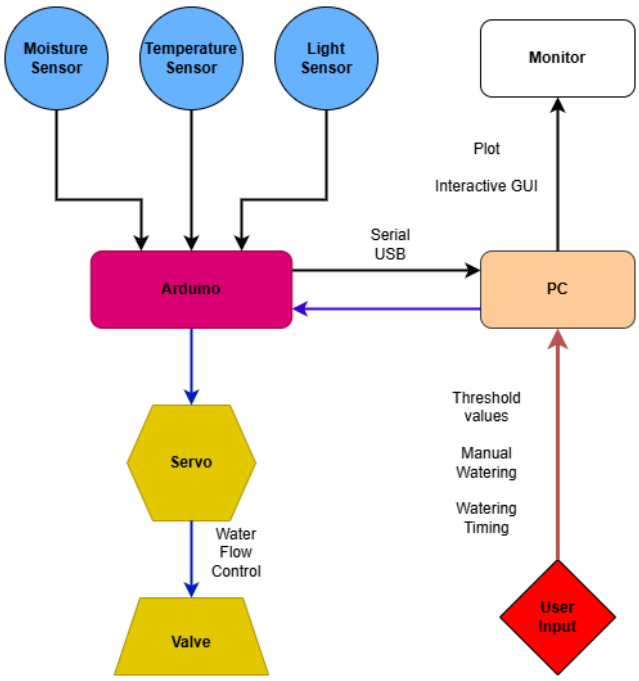

Figure 2: Block Diagram for the automatic watering system

D Firmware Code

D.1 firmware.cpp

```
#include <Servo.h>
//Pin Assignment as our schematic digram
#define ADC_VREF 5.0
#define ADC_RESOLUTION 1024.0
#define PIN_TMP36 A2
#define PIN_SOIL A5
#define SERVO_PIN 9

const int NUM_SAMPLES = 10;

Servo myservo;

void setup() {
  Serial.begin(9600);
  myservo.attach(SERVO_PIN);
}

float readAveragedAnalog(int pin) {
  long sum = 0;
  for (int i = 0; i < NUM_SAMPLES; i++) {
    sum += analogRead(pin);
    delay(5);  // small delay to allow ADC to settle
  }
  return sum / float(NUM_SAMPLES);
}

void loop() {
  // Average Sampling TMP36 and Soil moisture sensor
  float adcTemp = readAveragedAnalog(PIN_TMP36);
  float adcSoil = readAveragedAnalog(PIN_SOIL);

  // Conversion from TMP36 voltage ouput to temperature
  float voltage = adcTemp * (ADC_VREF / ADC_RESOLUTION);
  float tempC = (voltage - 0.5) * 100;

  // send temperature and moisture to pc end
  Serial.print("T=");
  Serial.print(tempC, 1);  // save first digit
  Serial.print(",M=");
  Serial.println((int)adcSoil);  // save as integer

  // recieve excuting signal form computer
  if (Serial.available()) {
    String cmd = Serial.readStringUntil('\n');
    cmd.trim();
    if (cmd.startsWith("SERVO=")) {
      int angle = cmd.substring(6).toInt();
      myservo.write(angle);
    }
  }

  delay(1000);  // every second
}
```

Code 1: Arduino code for sending data to the computer end

E Software Code

This software code is just a placeholder to simulate communication between the Arduino and the PC before the real system is functional.

E.1 Sender.py

```
# arduino_simulator.py
import socket
import time
import random

HOST = 'localhost'
PORT = 12345

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((HOST, PORT))
server.listen(1)
print("Arduino: waiting for connection...")
conn, addr = server.accept()
print("connected:", addr)

try:
    while True:
        temp = round(random.uniform(20.0, 30.0), 1)
        moisture = random.randint(300, 700)
        data = f"T:{temp} M:{moisture}\n"
        conn.sendall(data.encode('utf-8'))
        time.sleep(1)
except KeyboardInterrupt:
    print("END simulation")
finally:
    conn.close()
    server.close()
```

Code 2: Python script for sending data to the serial port

E.2 Receiver.py

```
# gui_test.py
import socket
import tkinter as tk
import threading

HOST = 'localhost'
PORT = 12345

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((HOST, PORT))

# GUI
root = tk.Tk()
root.title("Environment Monitoring - Simulation")

temperature_label = tk.Label(root, text="Temperature: -- C", font=("Arial", 16))
temperature_label.pack(pady=10)

moisture_label = tk.Label(root, text="Soil: --", font=("Arial", 16))
moisture_label.pack(pady=10)

def read_socket():
    buffer = ""
```

```
    while True:
        try:
            data = client.recv(1024).decode('utf-8')
            buffer += data
            while '\n' in buffer:
                line, buffer = buffer.split('\n', 1)
                if line.startswith("T:"):
                    parts = line.strip().split()
                    temp_str = parts[0][2:]
                    moisture_str = parts[1][2:]
                    temperature_label.config(text=f"Temperature: {temp_str} C")
                    moisture_label.config(text=f"Soil mositure: {moisture_str}")
        except Exception as e:
            print("Error:", e)
            break

threading.Thread(target=read_socket, daemon=True).start()
root.mainloop()
```

Code 3: Python script for receiving sending data to the serial port

References

[1] Arduino Documentation. *servo in Arduino*. Available at: https://docs.arduino.cc/learn/electronics/servo-motors/

[2] Analog Devices. *TMP35/TMP36/TMP37 Data Sheet*. Available at: https://www.analog.com/media/en/technical-documentation/data-sheets/TMP35_36_37.pdf

[3] Analog Devices. *TMP36 with Arduino*. Available at: https://arduinogetstarted.com/tutorials/arduino-tmp36-temperature-sensor

[4] DFRobot. *Capacitive Soil Moisture Sensor SKU:SEN0193*. Available at: https://wiki.dfrobot.com/Capacitive_Soil_Moisture_Sensor_SKU_SEN0193

[5] YouTube. *Improving Capacitive Soil Moisture Sensor Readings*, Available at: https://youtu.be/QGCrtXf8YSs

[6] Printables. *Pinch Valve Powered by Servo*. Available at: https://www.printables.com/model/247744-pinch-valve-powered-by-servo/files